PGI® Compilers and Tools
PGI Premier Support at ORNL

14 APR 09
XT5 Workshop - ORNL

Dave Norton
dave.norton@pgroup.com
530.544.9075
www.hpfa.com

Craig Toepfer
craig.toepfer@pgroup.com
503.682.2806
www.pgroup.com

Outline of Today's Topics

- What is PGI Premier Support?
- What is the motivation behind Premier Support
- Common Optimization Opportunities
- ORNL Premier Support work
- How can you take advantage of the PGI/ORNL relationship?
- Questions and Answers

What is PGI Premier Support?

• PGI Premier Support is a *professional services* program offered to select customers with the intent of direct engineer to engineer engagement on mission critical customer issues.

- Program components include:
  - PGI Quick Start Seminar with additional customized training options
  - A designated PGI technical contact within engineering
  - PGI Tracker online inquiry tracking system
  - Custom software patches and workarounds
  - Interim PGI releases to address critical issues
  - Custom libraries for runtime debugging
  - **Custom application performance analysis and tuning**
  - Custom compiler features

PGI/ORNL Classroom Training

PGI presented the Quick Start Seminar, an on-site ½ day introduction to PGI compilers and tools intended to cover best practices issues for getting code up and running optimally and giving correct results in the shortest amount of time, at ORNL in November

PGI offers additional training ranging from “hands-on interaction with code” sessions to in depth training on customer specific performance profiling, to customer specific application optimization, including assembly language seminars.

Customized training can be incorporated into the Premier Services program as desired by the customer.

If there is enough interest, we can present the Quick Start Seminar again.

PGI Premier Support Motivation

Customers – especially those with very large systems and specialty applications – have motivation to understand in detail the performance of their codes and *have a willingness to include compiler expertise directly on their code development teams*.

Code team members who specialize in the science of the application often do not have expertise in how a compiler views their application.

By adding a PGI compiler engineer to the application development team, the team gets access to in depth compiler knowledge, knowledge about how the compiler views code (or should view code) and therefore a team member who can help guide the code development process to optimize application performance while also working on the compiler so that is better understands the code.

Oak Ridge – Premier Support Example

Implemented PGI version of -finstrument_functions for James Rosinski. This is needed to support a tool that he has developed called "GPTL: A tool for characterizing parallel and serial application performance“

-Minstrument flag to the compiler

Generates instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions will be called with the address of the current function and its call site.

```
void __cyg_profile_func_enter (void *this_fn,
                               void *call_site);
void __cyg_profile_func_exit  (void *this_fn,
                               void *call_site);
```

.

.

Put 20 years of compiler expertise to work for *you*

Many scientists an engineers take a semester long course on compilers in college. Our engineers continued that study for the next 20 years.

They understand how to write code that the compiler can best ingest.

By examining the assembly code output, they understand when the compiler isn't generating as efficient code as it should.

There mission is to help you reach an optimal solution to writing, maintaining *and* getting maximal performance from your code.

Premier support is here for *you*!

What's New in PGI 8.0

- OpenMP 3.0 Support in Fortran and C (C++ in 8.1)
- Continued SPECFP06 and SPECINT06 Performance
- PGPROF improvements
- PGI Unified Binary enhancements
- Common Compiler Feedback Format
- Tuning for AMD Shanghai processors
- Accelerator Compiler Beta
- Improved C++ STL performance, features
- Bug fixes

Common Performance Challenges

Vectorization on both Intel and AMD processors

What is vectorization? Is my code vectorizing?

Conflicts with C++ and F90 "ease of use" programming techniques. C and C++ pointer issues that prevent vectorization.

Multi-core issues

Memory bandwidth

MPI, OpenMP, and auto parallelization

IPA – Interproceedural Analysis and Inlining

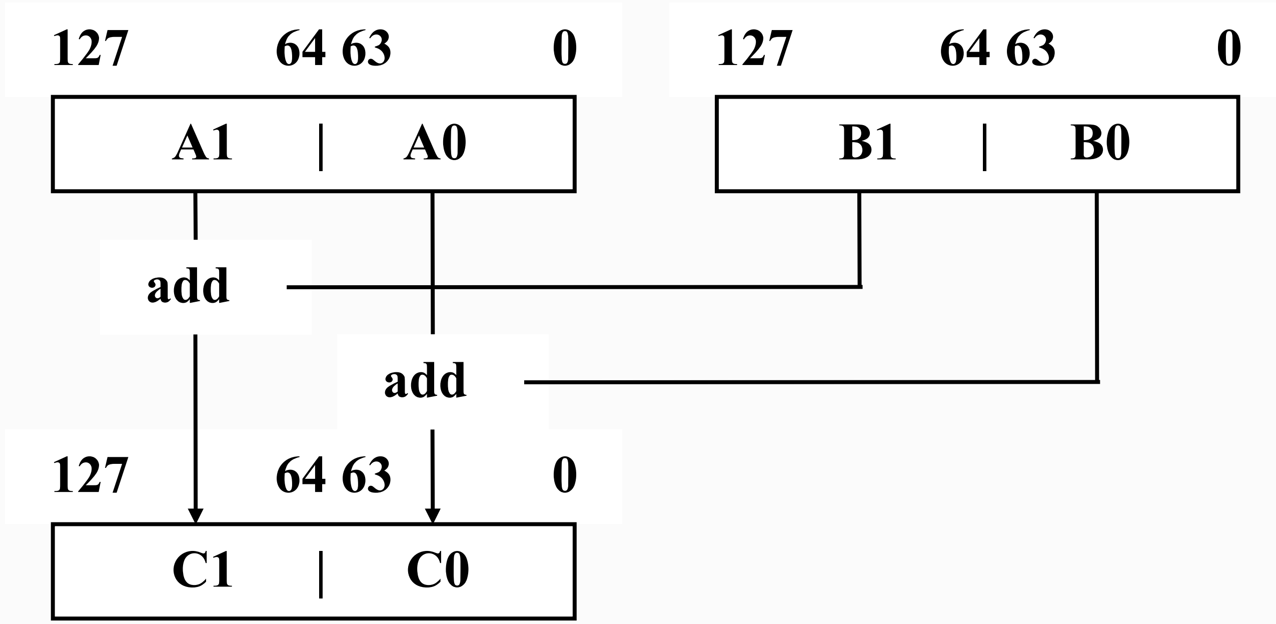
IPA and inline enabled libraries

What is Vectorization on x64 CPUs?

- **By a Programmer**: writing or modifying algorithms and loops to enable or maximize generation of x64 packed Streaming SIMD Extensions (SSE) instructions by a vectorizing compiler

- **By a Compiler**: identifying and transforming loops to use packed SSE arithmetic instructions which operate on more than one data element per instruction

Double-precision Packed SSE Operations on x64 CPUs



Double-precision Packed SSE *Implementations* on x64 CPUs

1st Gen

Cycle i: [A1|A0] [B1|B0]
\ /
A0+B0
.....
.....

Cycle i + 1: [A1|A0] [B1|B0]
\ /
A1+B1
A0+B0
.....

Cycle i + p - 1: .....
A1+B1
A0+B0
\
[.. |C0]

Cycle I + p: .....
.....
A1+B1
/
[C1|C0]

2nd Gen

Cycle i: [A1|A0] [B1|B0]
\ /
[A1+B1|A0+B0]
.....
.....

Cycle i + 1: .....
[A1+B1|A0+B0]
.....

Cycle i + p: .....
.....
[A1+B1|A0+B0]
\ /
[C1|C0]

Vectorizable Loop in SPECFP2K FACEREC Data is REAL*4

```
350 !
351 !   Initialize vertex, similarity and coordinate arrays
352 !
353     Do Index = 1, NodeCount
354        IX = MOD (Index - 1, NodesX) + 1
355        IY = ((Index - 1) / NodesX) + 1
356        CoordX (IX, IY) = Position (1) + (IX - 1) * StepX
357        CoordY (IX, IY) = Position (2) + (IY - 1) * StepY
358        JetSim (Index)  = SUM (Graph (:, :, Index) * &
359     &                    GaborTrafo (:, :, CoordX(IX,IY), CoordY(IX,IY)))
360        VertexX (Index) = MOD (Params%Graph%RandomIndex (Index) - 1, NodesX) + 1
361        VertexY (Index) = ((Params%Graph%RandomIndex (Index) - 1) / NodesX) + 1
362     End Do
```

Inner loop at line 358 is vectorizable, can used packed SSE instructions

Use –Minfo to see Which Loops Vectorize

```
% pgf95 -fast -Minfo  graphRoutines.f90
…
localmove:
   334, Loop unrolled 1 times (completely unrolled)
   343, Loop unrolled 2 times (completely unrolled)
   358, Generating vector sse code for inner loop
   364, Generating vector sse code for inner loop
        Generating vector sse code for inner loop
   392, Generating vector sse code for inner loop
   423, Generating vector sse code for inner loop
%
```

`-fast` Includes "`-Mvect=sse -Mcache_align -Mnoframe -Mlre`"

Scalar SSE:

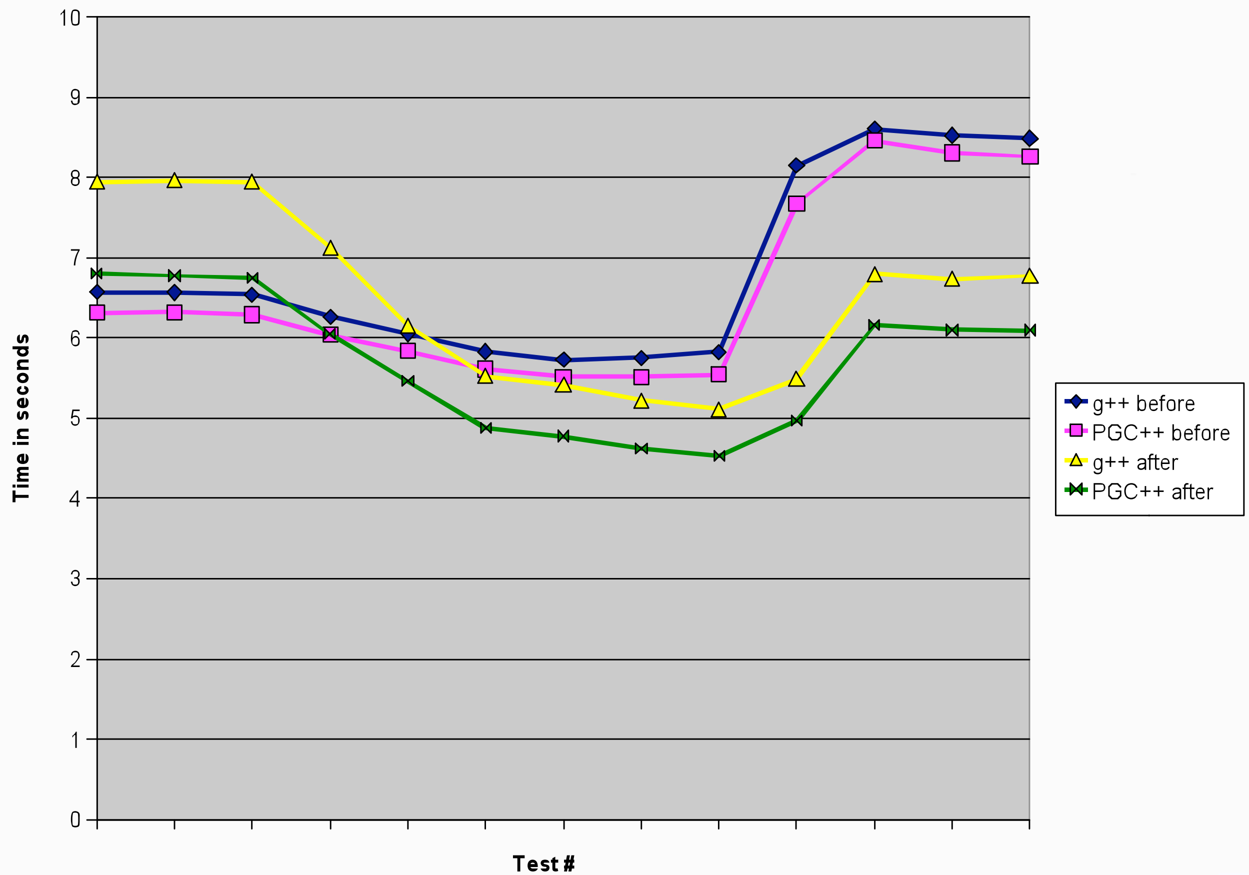
```
.LB6_668:
# lineno: 358
        movss   -12(%rax),%xmm2
        movss   -4(%rax),%xmm3
        subl    $1,%edx
        mulss   -12(%rcx),%xmm2
        addss   %xmm0,%xmm2
        mulss   -4(%rcx),%xmm3
        movss   -8(%rax),%xmm0
        mulss   -8(%rcx),%xmm0
        addss   %xmm0,%xmm2
        movss   (%rax),%xmm0
        addq    $16,%rax
        addss   %xmm3,%xmm2
        mulss   (%rcx),%xmm0
        addq    $16,%rcx
        testl   %edx,%edx
        addss   %xmm0,%xmm2
        movaps  %xmm2,%xmm0
        jg      .LB6_625
```

Vector SSE:

```
.LB6_1245:
# lineno: 358
        movlps  (%rdx,%rcx),%xmm2
        subl    $8,%eax
        movlps  16(%rcx,%rdx),%xmm3
        prefetcht0  64(%rcx,%rsi)
        prefetcht0  64(%rcx,%rdx)
        movhps  8(%rcx,%rdx),%xmm2
        mulps   (%rsi,%rcx),%xmm2
        movhps  24(%rcx,%rdx),%xmm3
        addps   %xmm2,%xmm0
        mulps   16(%rcx,%rsi),%xmm3
        addq    $32,%rcx
        testl   %eax,%eax
        addps   %xmm3,%xmm0
        jg      .LB6_1245:
```

Facerec Scalar: 104.2 sec
Facerec Vector: 84.3 sec

Benefits of Vectorization - Intel Core 2 Alegra Kernel Performance



The Portland Group

PGI and Oak Ridge - Vectorization

- For all of our fastmath library intrinsics, we created barcelona-tuned versions for ORNL. This involved manually converting assembly movlpd to movsd, a few other ops like movddup instead of a movlpd/movhpd pair when appropriate. Created two entry points so both versions could exist in a PGI unified binary. And then the appropriate compiler changes to call the correct one. This ended up speeding up sine and cosine by about 30% on a barcelona.
- Created vector LOG10. This allowed some key loops in S3D to vectorize thus enabling a significant speed up in the application.

Common Barriers to SSE Vectorization

- **Potential Dependencies & C Pointers** – Give compiler more info with –Msafeptr, pragmas, or **restrict** type qualifer
- **Function Calls** – Try inlining with –Minline or –Mipa=inline
- **Type conversions** – manually convert constants or use flags
- **Large Number of Statements** – Try –Mvect=nosizelimit
- **Too few iterations** – Usually better to unroll the loop
- **Real dependencies** – Must restructure loop, if possible

Vectorizable C Code Fragment?

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222       u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224       float vdt = v[i] * dt;
225       u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Minfo –Mneginfo functions.c
func4:
    221, Loop unrolled 4 times
    221, Loop not vectorized due to data dependency
    223, Loop not vectorized due to data dependency
```

Pointer Arguments Inhibit Vectorization

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222       u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224       float vdt = v[i] * dt;
225       u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Msafeptr –Minfo functions.c
func4:
    221, Generated vector SSE code for inner loop
         Generated 3 prefetch instructions for this loop
    223, Unrolled inner loop 4 times
```

C Constant Inhibits Vectorization

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222       u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224       float vdt = v[i] * dt;
225       u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Msafeptr –Mfcon –Minfo functions.c
func4:
    221, Generated vector SSE code for inner loop
         Generated 3 prefetch instructions for this loop
    223, Generated vector SSE code for inner loop
         Generated 4 prefetch instructions for this loop
```

-Msafeptr vs.Pragma vs. restrict (sledgehammer vs. scalpel)

–M[no]safeptr[=all | arg | auto | dummy | local | static | global]

all All pointers are safe

arg Argument pointers are safe

local local pointers are safe

static static local pointers are safe

global global pointers are safe

#pragma [*scope*] [no]safeptr={arg | local | global | static | all},…

Where *scope* is *global*, *routine* or *loop*

The Portland Group

3 Steps to Multi-core Performance

1) Optimize single-core performance

2) Enable multi-core auto-parallelization

3) Tune for multi-core with OpenMP directive-based parallelization

1. Optimize Single-core Performance

PGI compilers give realtime optimization hints, and support many tuning options and directives

See http://www.pgroup.com/lit/pgi_article_tuning.pdf for generic tuning tips

See http://www.pgroup.com/lit/pgi_article_CUG07.pdf for C++ tuning tips

Use PGPROF or other profilers to reveal single-core performance issues

Conflicts with C++ and F90 “ease of use” programming techniques. C and C++ pointer issues that prevent vectorization.

Modern programming techniques in C++ and occasionally in object oriented F90 code lead to levels of code obfuscation that the compiler simply is unable to unwind.

Alegra – C++ Challenges (con't)

For this dataset, the value of mat_max is 21, and the number of element blocks(mesh->Num_Element_Blocks()) is 1. The LOCAL_ELEMENT_LOOP is excuted 160000 times.

Using the debugger, the three lines of code:

1) **Real volume_old = el->Volume_Fraction(m);**

The assembly instructions generated for this line of code dereferenced memory 4 times as follows:

```
movq    160(%rcx), %rdx      <--- Address of el.material_data
movq    (%rdx,%rax,8), %rsi  <--- Address of el.material_data[m].material
movq    40(%rsi), %rax       <--- Address of el.material_data[m].material.data[m']
movl    (%rax), %xmm0        <--- Value of volume_old
```

Alegra – C++ Challenges (con't)

2) **Real* scratch = el->Scalar_Array(REMAP_SCRATCH);**

The assembly instructions generated for this line of code dereferenced memory 2 times as follows:

```
movq    8(%rdx,%rcx), %rsi   <--- Address of el.data
leaq    (%rsi,%rax,8), %rdi  <--- Address of el.data+m
```

3) **Material_Data* pmat_data = el->Material_Data_Ptr(m);**

The assembly instructions generated for this line of code dereferenced memory 2 times as follows:

```
movq    160(%rcx), %r9       <--- Address of el.material_data
movq    (%r9,%r8,8), %rax    <--- Address of el.material_data[m]
```

Remedies for Alegra pointer issues

- Short circuiting the pointer chain from one time step to the next

- Rearrange the location of critical element in the data structure so that element data is in cache when data structure is first referenced

PGI and Oak Ridge – Listing Files

- ORNL requested a feature which combined the output of compiler messages and program files.
  - Users can get information about what the optimizations that compiler makes, and the reasons it is unable to make other optimizations by using the compile time flags:

-Minfo and –Mneginfo

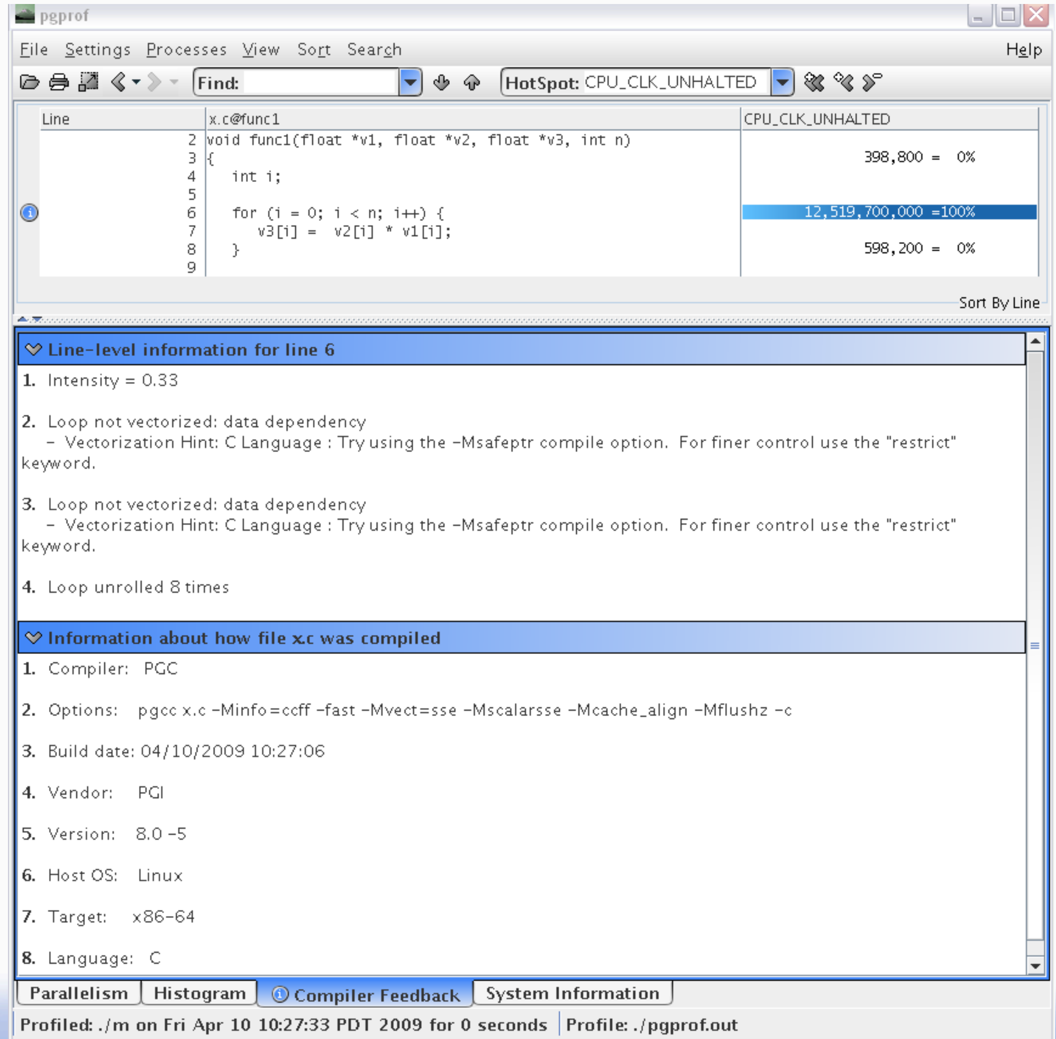
  - At the request of ORNL, users can now see these messages along with the code that is is causing the messages by using the -Minfo=ccff

Demo: Initial Build/Run

toepfer@grandcanary:~/ORNL/ccff

```
toepfer@grandcanary:~/ORNL/ccff> make m
pgcc -Minfo=ccff -fast -c m.c
pgcc -Minfo=ccff -fast -c x.c
as -o dclock.o dclock.s
pgcc -Minfo=ccff -fast -o m m.o x.o dclock.o
toepfer@grandcanary:~/ORNL/ccff> make run
taskset 0x40 /bin/time pgcollect -exe ./m
Using 2.6+ OProfile kernel interface.
Using log file /var/lib/oprofile/samples/oprofiled.log
Daemon started.
Profiler running.
v3....  1.000000
Total Time(func1):  6.346310
v3....  3.000000
Total Time(func2):  16.118524
Stopping profiling.
Killing daemon.
22.94user 0.51system 0:25.59elapsed 91%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+153903minor)pagefaults 0swaps
toepfer@grandcanary:~/ORNL/ccff>
```

Demo: Profile of Initial Run

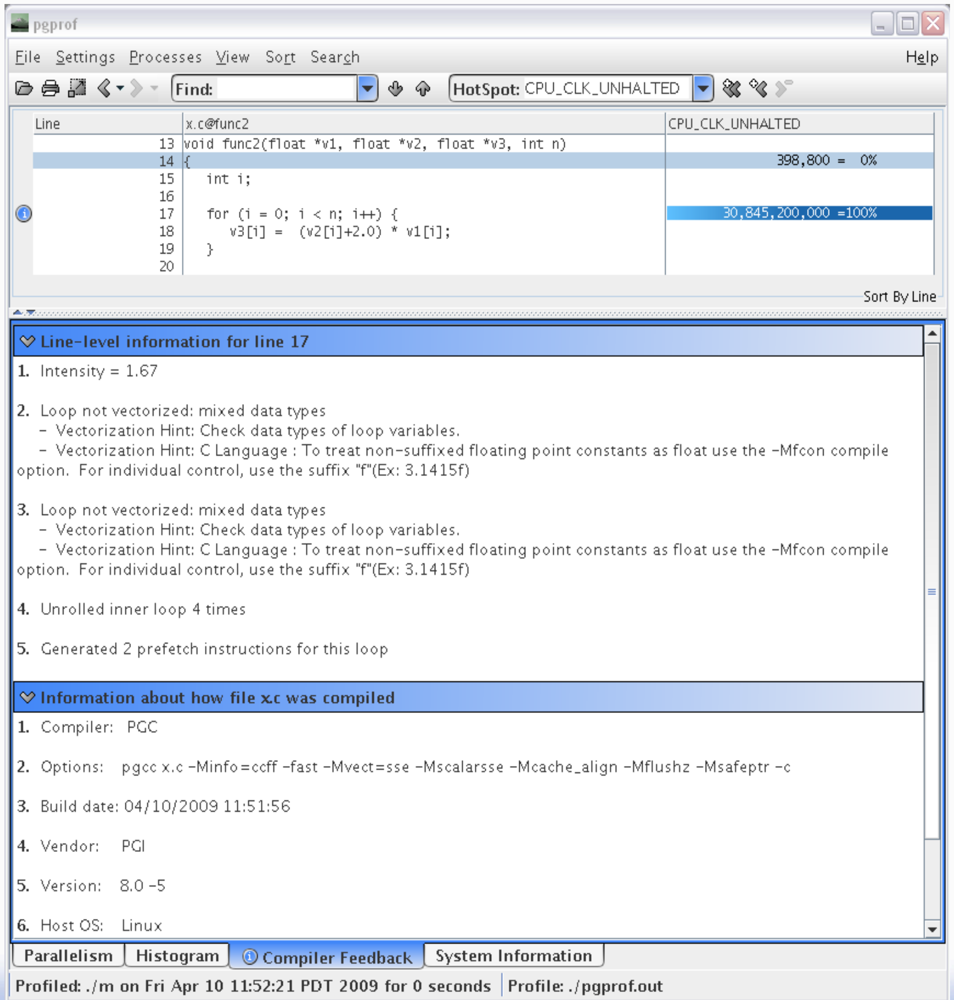


Demo: Build/Run with -Msafeptr

toepfer@grandcanary:~/ORNL/ccff

```
toepfer@grandcanary:~/ORNL/ccff> make m
pgcc -Minfo=ccff -fast -Msafeptr -c m.c
pgcc -Minfo=ccff -fast -Msafeptr -c x.c
as -o dclock.o dclock.s
pgcc -Minfo=ccff -fast -Msafeptr -o m m.o x.o dclock.o
toepfer@grandcanary:~/ORNL/ccff> make run
taskset 0x40 /bin/time pgcollect -exe ./m
Using 2.6+ OProfile kernel interface.
Using log file /var/lib/oprofile/samples/oprofiled.log
Daemon started.
Profiler running.
v3....  1.000000
Total Time(func1):  3.945711
v3....  3.000000
Total Time(func2):  15.327262
Stopping profiling.
Killing daemon.
19.76user 0.49system 0:22.38elapsed 90%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+153888minor)pagefaults 0swaps
toepfer@grandcanary:~/ORNL/ccff>
```

Demo: Profile of –Msafeptr Run

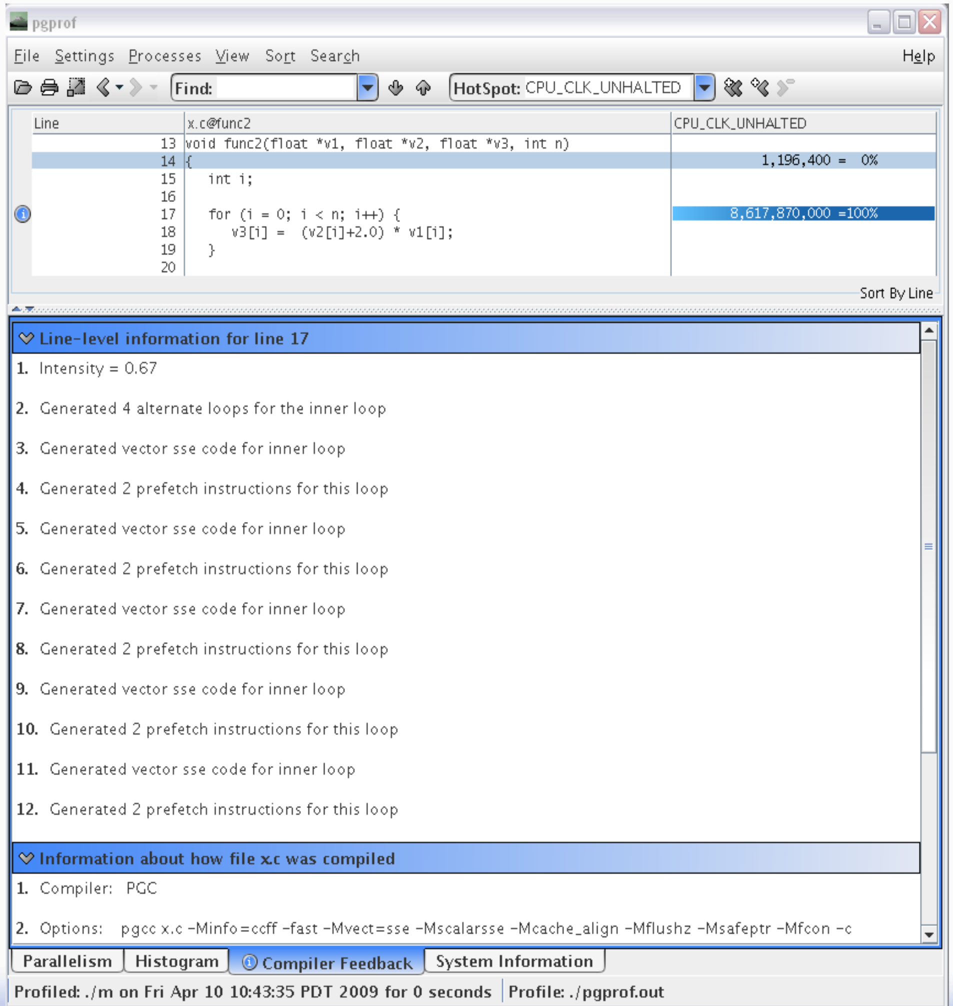


Demo: Build/Run with -Mfcon

toepfer@grandcanary:~/ORNL/ccff

```
toepfer@grandcanary:~/ORNL/ccff> make m
pgcc -Minfo=ccff -fast -Msafeptr -Mfcon -c m.c
pgcc -Minfo=ccff -fast -Msafeptr -Mfcon -c x.c
as -o dclock.o dclock.s
pgcc -Minfo=ccff -fast -Msafeptr -Mfcon -o m m.o x.o dclock.o
toepfer@grandcanary:~/ORNL/ccff> make run
taskset 0x40 /bin/time pgcollect -exe ./m
Using 2.6+ OProfile kernel interface.
Using log file /var/lib/oprofile/samples/oprofiled.log
Daemon started.
Profiler running.
v3....  1.000000
Total Time(func1):  3.984802
v3....  3.000000
Total Time(func2):  4.370197
Stopping profiling.
Killing daemon.
8.80user 0.52system 0:11.44elapsed 81%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+153882minor)pagefaults 0swaps
toepfer@grandcanary:~/ORNL/ccff>
```

Demo: Profile of –Mfcon Run



2. Enable Multi-core Auto-parallelization

`-Mconcur[=`*`option`*`[,`*`option`*`]]` where *`option`* is:

`[no]altcode:`<**n**>	[Don't] Generate alternate serial code for parallel loops
`dist:block`	Parallelize with block distribution (default)
`dist:cyclic`	Parallelize with cyclic distribution
`cncall`	Loops with calls are candidates for parallelization
`[no]assoc`	Disable/Enable parallelization of loops with reductions
`innermost`	Enable parallelization of innermost loops
`levels:`<**n**>	Parallelize loops nested at most **n** levels deep
bind	Bind threads to cores. Must be specified at link time

SPEC OMP2001 314.MGRID_M

3D Multigrid Solver

```
% pgf95 –Mconcur –Minfo –fast  mgrid.f
…
resid:
    366, Parallel code for non-innermost loop activated
              if loop count >= 33; block distribution
    368, 4 loop-carried redundant expressions removed
              with 12 operations and 16 arrays
          Generated 4 alternate loops for the inner loop
          Generated vector SSE code for inner loop
          Generated 8 prefetch instructions for this loop
          Generated vector SSE code for inner loop
          Generated 8 prefetch instructions for this loop
```

314.MGRID_M Benchmark Main Loop

```
      DO 10 I3=2, N-1
      DO 10 I2=2,N-1
      DO 10 I1=2,N-1
10       R(I1,I2,I3) = V(I1,I2,I3)
     &            -A(0)*(U(I1,I2,I3))
     &            -A(1)*(U(I1-1,I2,I3)+U(I1+1,I2,I3)
     &                  +U(I1,I2-1,I3)+U(I1,I2+1,I3)
     &                  +U(I1,I2,I3-1)+U(I1,I2,I3+1))
     &            -A(2)*(U(I1-1,I2-1,I3)+U(I1+1,I2-1,I3)
     &                  +U(I1-1,I2+1,I3)+U(I1+1,I2+1,I3)
     &                  +U(I1,I2-1,I3-1)+U(I1,I2+1,I3-1)
     &                  +U(I1,I2-1,I3+1)+U(I1,I2+1,I3+1)
     &                  +U(I1-1,I2,I3-1)+U(I1-1,I2,I3+1)
     &                  +U(I1+1,I2,I3-1)+U(I1+1,I2,I3+1) )
     &            -A(3)*(U(I1-1,I2-1,I3-1)+U(I1+1,I2-1,I3-1)
     &                  +U(I1-1,I2+1,I3-1)+U(I1+1,I2+1,I3-1)
     &                  +U(I1-1,I2-1,I3+1)+U(I1+1,I2-1,I3+1)
     &                  +U(I1-1,I2+1,I3+1)+U(I1+1,I2+1,I3+1))
```

Auto-parallel 314.MGRID_M Runtime on Single Socket Quad-core AMD Opteron

NUM_THREADS	Runtime (seconds)	Speed-up
1	208	
2	116	1.8
4	100	2.1

The Portland Group

3. Tune for Multi-core with OpenMP

- PGI supports full native OpenMP 3.0 parallel programming directives/pragmas/runtime for F95 & C
  - C++ support in next major release
- PGI provides environment variables to maximize OpenMP performance (thread scheduling, binding, etc)
- Use –mp option to enable OpenMP compilation
- OpenMP programs compiled w/out –mp “just work”
- –Mconcur and –mp can be used together

SPEC OMP2001 314.MGRID_M

3D Multigrid Solver

```
% pgf95 –mp –Minfo –fastsse  mgrid.f
…
resid:
    360, Parallel region activated
    366, Parallel loop activated with static block schedule
    368, 4 loop-carried redundant expressions removed
              with 12 operations and 16 arrays
         Generated 4 alternate loops for the inner loop
         Generated vector SSE code for inner loop
         Generated 8 prefetch instructions for this loop
         …
     382, Parallel region terminated
```

OpenMP 314.MGRID_M Runtime on Single Socket Quad-core AMD Opteron

NUM_THREADS	Runtime (seconds)	Speed-up
1	205	
2	113	1.8
4	97	2.1

1 Step to Multi-core/Multi-Socket Performance?

1) Increase the thread count and rerun the application…

```
% export OMP_NUM_THREADS=8
% a.out
```

314.MGRID_M Runtime on Dual Socket Quad-core AMD Opteron

NUM_THREADS	Auto-Par executable Runtime(seconds)	OpenMP executable Runtime(seconds)
1	208	205
2	116	113
4	100	97
8	92	90

Multi-Socket Introduces Potential NUMA Issues

Physical memory location based on first touch

Use /usr/bin/numactl –hardware to identify potential memory hotspots while application is running

Minimize data initialization in serial regions of code

Modified 314.MGRID_M Runtime on Dual Socket Quad-core AMD Opteron

NUM_THREADS	Auto-Par executable Runtime(seconds)	OpenMP executable Runtime(seconds)
1	208	205
2	116	113
4	100	97
8	55	53

Multi-core Performance Guidelines

- Use correct target processor, -tp barcelona-64
- Vectorization and single core performance is a good start
- Compiler Feedback: a positive force in HPC SW Evolution
- The PGI -Mconcur flag can handle simple cases, and might surprise you with where it can find parallelism
- OpenMP gives finer control, is supported everywhere
- Gather some profiling data on where cache misses or other delays occur

Multi-core Performance Guidelines

- Design algorithms that minimize data movement and maximize data movement efficiency, rather than minimizing computations. FLOPS are free, bandwidth is precious.

- Strip-mining or other caching techniques (tiling, blocking) can be important.

- Use directives/pragmas/compiler options for fine-tuned control over memory-tuning optimization.

- Pay attention to NUMA effects when running on multi-socket nodes. Control location of thread execution using environment variables.

What can Interprocedural Analysis and Optimization with –Mipa do for You?

- Interprocedural constant propagation
- Pointer disambiguation
- Alignment detection, Alignment propagation
- Global variable mod/ref detection
- F90 shape propagation
- Function inlining
- IPA optimization of libraries, including inlining

Effect of IPA on the WUPWISE Benchmark

PGF95 Compiler Options	Execution Time in Seconds
–fast	156.49
–fast –Mipa=fast	121.65
–fast –Mipa=fast,inline	91.72

- –Mipa=fast => constant propagation => compiler sees complex matrices are all 4x3 => completely unrolls loops
- –Mipa=fast,inline => small matrix multiplies are all inlined

Using Interprocedural Analysis

- Must be used at both compile time and link time
- Non-disruptive to development process – edit/build/run
- Speed-ups of 5% - 10% are common
- –Mipa=safe:<*name*> - safe to optimize functions which call or are called from unknown function/library *name*
- –Mipa=libopt – perform IPA optimizations on libraries
- –Mipa=libinline – perform IPA inlining from libraries

Other C++ recommendations

- **Encapsulation, Data Hiding -** small functions, inline!
- **Exception Handling** – use –no_exceptions until 7.0
- **Overloaded operators, overloaded functions -** okay
- **Pointer Chasing -** -Msafeptr, restrict qualifer, 32 bits?
- **Templates, Generic Programming** – now okay
- **Inheritance, polymorphism, virtual functions** – runtime lookup or check, no inlining, potential performance penalties

Explicit Function Inlining

–Minline[=[lib:]<inlib> | [name:]<func> | except:<func> | size:<n> | levels:<n>]

[lib:]<inlib> Inline extracted functions from *inlib*

[name:]<func> Inline function func

except:<func> Do not inline function func

size:<n> Inline only functions smaller than n statements (approximate)

levels:<n> Inline n levels of functions

For C++ Codes, PGI Recommends IPA-based inlining or –Minline=levels:10!

Miscellaneous Optimizations (1)

- –Mfprelaxed – single-precision sqrt, rsqrt, div performed using reduced-precision reciprocal approximation
- –lacml and –lacml_mp – link in the AMD Core Math Library
- –Mprefetch=d:<p>,n:<q> – control prefetching distance, max number of prefetch instructions per loop
- –tp k8-32 – can result in big performance win on some C/C++ codes that don't require > 2GB addressing; pointer and long data become 32-bits

Miscellaneous Optimizations (2)

- –O3 – more aggressive hoisting and scalar replacement; not part of –fastsse, always time your code to make sure it's faster
- For C++ codes: —no_exceptions –zc_eh
- –M[no]movnt – disable / force non-temporal moves
- –V[version] to switch between PGI releases at file level
- –Mvect=noaltcode – disable multiple versions of loops

PGI Premier Conclusions

- Delivers education to better use compilers and tools
- Provides direct scientist to engineer interaction
- Provides custom compiler and library work
- Provides a compiler engineer for your code development team
- Results in faster application results!

Take advantage of *your* Premier Support opportunities!